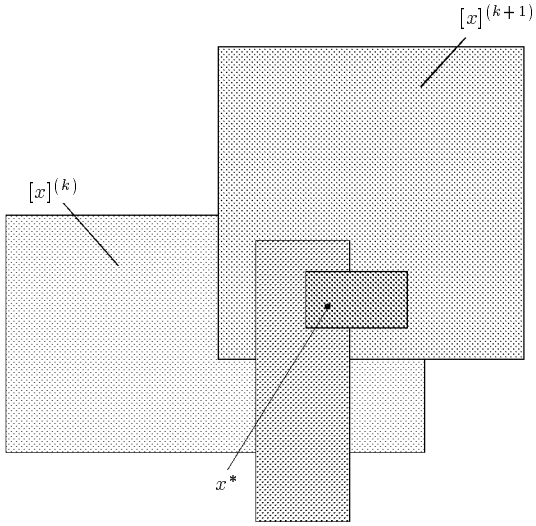
Institut für
Angewandte
Mathematik

Universität Karlsruhe (TH)
D-76128 Karlsruhe

An Optimized Interval Slope Arithmetic and its Application

Dietmar Ratz

**Forschungsschwerpunkt
Computerarithmetik,
Intervallrechnung und
Numerische Algorithmen mit
Ergebnisverifikation**



Bericht 4/1996

Impressum

Herausgeber:	Institut für Angewandte Mathematik Lehrstuhl Prof. Dr. Ulrich Kulisch Universität Karlsruhe (TH) D-76128 Karlsruhe
Redaktion:	Dr. Dietmar Ratz

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über

`ftp://iamk4515.mathematik.uni-karlsruhe.de`

im Verzeichnis: `/pub/documents/reports`

oder über die World Wide Web Seiten des Instituts

`http://www.uni-karlsruhe.de/~iam`

Autoren-Kontaktadresse

Rückfragen zum Inhalt dieses Berichts bitte an

Dr. Dietmar Ratz
Institut für Angewandte Mathematik
Universität Karlsruhe (TH)
D-76128 Karlsruhe

E-Mail: Dietmar.Ratz@math.uni-karlsruhe.de

An Optimized Interval Slope Arithmetic and its Application

Dietmar Ratz

Contents

Zusammenfassung

Eine optimierte Intervallsteigungsarithmetik und ihre Anwendung: In dieser Arbeit beschreiben wir eine Intervallsteigungsarithmetik, die im Hinblick auf die Berechnung von engeren Einschließungen für die tatsächlichen Steigungswerte optimiert ist. Ermöglicht wird dies durch die Verwendung spezieller Formeln für konvexe/konkave bzw. lokal konvexe/konkave Elementarfunktionen, die die üblicherweise verwendeten Ableitungswerte ersetzen. Wir behandeln die Details einer praktischen Realisierung einer solchen Intervallsteigungsarithmetik sowie deren Implementierung.

Außerdem zeigen wir wie diese Intervallsteigungsarithmetik erfolgreich auf dem Gebiet der eindimensionalen globalen Optimierung angewendet werden kann. Wir beschreiben die praktischen Aspekte einer neuen Pruning-Technik auf der Basis von Steigungen im Kontext von Intervall-Branch-and-Bound-Methoden und zeigen, daß es möglich ist, den häufig verwendeten Monotonie-Test durch einen Pruning-Schritt zu ersetzen. Dies führt zu einer beträchtlichen Verbesserung der Effizienz des globalen Optimierungsverfahrens.

Abstract

An Optimized Interval Slope Arithmetic and its Application: We describe an interval slope arithmetic which is optimized in view of the possibility to compute sharper enclosures of the actual slope values. For this purpose, special formulas are used for convex/concave or locally convex/concave elementary functions to replace the normally used derivative values. We treat the details of a practical realization and we give an implementation of the slope arithmetic.

Moreover, we show how our interval slope arithmetic can succesfully be applied in the field of one-dimensional global optimization. We describe the practical aspects of a new pruning technique based on slopes in the context of interval branch-and-bound methods. We show, that it is possible to replace the frequently used monotonicity test by a new pruning step which provides considerable improvement in efficiency of the global optimization method.

1 Introduction and Notation

Many interval methods use interval slopes (together with centered forms) to achieve better enclosures for the function ranges than those achieved with derivatives, as described in [8] and [6], for example. Very often, interval slopes are computed via an automatic differentiation process (eg. by using an interval slope arithmetic in connection with operator overloading). Within that process, interval extensions for the slopes of elementary functions are needed, and often interval evaluations of the derivatives are used for this purpose (cf. [8]).

In this paper, we deal with the practical realization and implementation of an interval slope arithmetic, which is optimized in the sense that we use alternative formulas for the slopes of elementary functions to achieve better enclosures. Additionally, we demonstrate the application of our slope arithmetic in a global optimization method equipped with a new pruning technique described in [12].

In the following, we denote real numbers by $x, y, \ldots$ and real bounded and closed intervals by $X = [\underline{x}, \overline{x}] = [\inf(X), \sup(X)], Y = [\underline{y}, \overline{y}] = [\inf(Y), \sup(Y)], \ldots$.

The set of compact intervals is denoted by $I\!I\!R := \{[a, b] \mid a \leq b,\ a, b \in I\!R\}$. The width or diameter of the interval X is defined by $d(X) = \overline{x} - \underline{x}$, and the midpoint of the interval X is defined by $m(X) = (\underline{x} + \overline{x})/2$.

We call a function $F : I\mathbb{R} \to I\mathbb{R}$ an *inclusion function* of $f : \mathbb{R} \to \mathbb{R}$ in X, if $x \in X$ implies $f(x) \in F(X)$. In other words, $f_{\mathrm{rg}}(X) \subseteq F(X)$, where $f_{\mathrm{rg}}(X)$ is the range of the function f on X. The inclusion function of the derivative of f is denoted by F'. Inclusion functions can be computed via interval arithmetic [1, 3] for almost all functions specified by a finite algorithm (i.e. not only for given expressions). Moreover applying so-called automatic differentiation or differentiation arithmetic in connection with interval arithmetic [3, 4, 6], we are also able to compute the inclusion function for the derivatives or the slopes.

Automatic differentiation combines the advantages of symbolic and numerical differentiation and handles numbers instead of symbolic formulas. The computation of the derivative (or slope) is done automatically together with the computation of the function value. The main advantage of this process is that only the algorithm or formula for the function is required. No explicit formulas for the derivative (or slope) is required.

It is assumed in the following that the inclusion functions have the *isotonicity* property, i.e. $X \subseteq Y$ implies $F(X) \subseteq F(Y)$.

2 Centered Forms and Interval Slopes

Centered forms (see [1, 6, 8]) are special interval extensions and serve to reduce the overestimation in computing interval enclosures of the range of a function f over some interval X. Usually, a centered form is derived from the mean-value theorem. Suppose f is differentiable on its domain D. Then $f(x) = f(c) + f'(\xi)(x - c)$ with some fixed $c \in D$ and ξ between x and c. Let $c, x \in X$, so $\xi \in X$. Therefore

$$\begin{aligned} f(x) \;=\; f(c) + f'(\xi)(x-c) \;&\in\; f(c) + f'(X) \cdot (x-c) \\ &\subseteq\; f(c) + F'(X) \cdot (X-c). \end{aligned} \tag{1}$$

The latter is called *centered form* or *generalized mean value form* of f over X. Here, f is extended with respect to every $x \in X$, since $G = F'(X)$ is an interval evaluation of the derivative of f over the entire interval X.

Krawczyk and Neumaier [8] showed that if we have an interval $S \in I\mathbb{R}$ such that, for all $x \in X$ we have

$$f(x) = f(c) + s \cdot (x - c) \quad \text{for some } s \in S, \tag{2}$$

then the interval $F_s(X) := f(c) + S \cdot (X - c)$ encloses the range of f over X, that is $f_{\mathrm{rg}}(X) \subseteq F_s(X)$. Such an interval S can be calculated by means of an interval slope and not only with an interval derivative. If we use a slope, then f is extended with respect to an arbitrary but fixed $c \in X$.

Definition 2.1 *The function $s_f : D \times D \to \mathbb{R}$ with*

$$f(x) = f(c) + s_f(c, x) \cdot (x - c)$$

is called a slope *(between c and x). In the one-dimensional case ($D \subseteq \mathbb{R}$), we have*

$$s_f(c,x) = \begin{cases} \dfrac{f(x)-f(c)}{x-c} & \text{if } x \neq c \\ \widetilde{s} & \text{if } x = c, \end{cases}$$

where $\widetilde{s} \in \mathbb{R}$ may be arbitrarily chosen. Assuming f to be differentiable and the slope to be continuous, we can define $\widetilde{s} := f'(c)$.

Moreover, we define the interval slope *of f over the interval X by*

$$s_f(c,X) := \{s_f(c,x) \mid x \in X, \quad x \neq c\},$$

where it is not necessary that f is differentiable.

Remarks: (i) It is easy to see that $S = s_f(c,X)$ satisfies (2) and

$$f(x) \in f(c) + S \cdot (x-c) \subseteq f(c) + S \cdot (X-c). \tag{3}$$

(ii) Often $c = m(X)$ is used to compute the interval slope.
(iii) If we assume f to be continuously differentiable, then we have (cf. [8])

$$s_f(c,X) \subseteq s_f(X,X) = F'(X). \tag{4}$$

Slopes as well as interval slopes can be calculated by means of an automatic differentiation process ([3], [6], [8]). The main advantage of this process is that only the algorithm or formula for the function is required. No explicit formulas for the derivatives or slopes are required.

3 An Optimized Interval Slope Arithmetic

Automatic differentiation for slopes evaluates functions specified by algorithms or formulas, where all operations are executed according to the rules of a *slope arithmetic*, which is an arithmetic for ordered triples of the form

$$\mathcal{U} = (U_x, U_c, U_s), \quad \text{with } U_x, U_c, U_s \in I\mathbb{R}.$$

Definition 3.1 *The triple $\mathcal{U} = (U_x, U_c, U_s)$ with $U_x, U_c, U_s \in I\mathbb{R}$ is called* a slope triple *for a function $u : D \to \mathbb{R}$ with $D \subseteq \mathbb{R}$, an interval $X \in I\mathbb{R}$ ($X \subseteq D$) and a fixed $c \in X$, if*

$$u(x) \in U_x, \tag{5}$$
$$u(c) \in U_c \quad \text{and} \tag{6}$$
$$u(x) - u(c) \in U_s \cdot (x-c) \tag{7}$$

for all $x \in X$.

It is easy to see how slope triples for a constant function or the identity function (representing the independent variable x in our slope arithmetic) must be defined:

Lemma 3.2 *$\mathcal{C} = (\lambda, \lambda, 0)$ is a slope triple for the function $u(x) \equiv \lambda \in \mathbb{R}$, and $\mathcal{X} = (X, c, 1)$ with $c \in X$ is a slope triple for the function $u(x) \equiv x$.*

Proof: Trivial. □

The rules for the *slope arithmetic* can be fixed as:

$$\mathcal{W} = \mathcal{U} \pm \mathcal{V} \Rightarrow \begin{cases} W_x = U_x \pm V_x, \\ W_c = U_c \pm V_c, \\ W_s = U_s \pm V_s, \end{cases} \tag{8}$$

$$\mathcal{W} = \mathcal{U} \cdot \mathcal{V} \Rightarrow \begin{cases} W_x = U_x \cdot V_x, \\ W_c = U_c \cdot V_c, \\ W_s = U_x \cdot V_s + U_s \cdot V_c, \end{cases} \tag{9}$$

$$\mathcal{W} = \mathcal{U} / \mathcal{V} \Rightarrow \begin{cases} W_x = U_x / V_x, \\ W_c = U_c / V_c, \\ W_s = (U_s - W_c \cdot V_s)/V_x, \end{cases} \tag{10}$$

where $0 \notin V_x$ is assumed in case of division.

For an elementary function φ and $\mathcal{U} = (U_x, U_c, U_s)$, we can fix:

$$\mathcal{W} = \varphi(\mathcal{U}) \Rightarrow \begin{cases} W_x = \varphi(U_x), \\ W_c = \varphi(U_c), \\ W_s = U_s \cdot s_\varphi(U_c, U_x). \end{cases} \tag{11}$$

For these rules we can state the following theorem, which is very similar to Theorem 2.3.8 and Proposition 2.3.9 in [8]:

Theorem 3.3 *Let $\mathcal{U}$ and $\mathcal{V}$ be slope triples of the functions $u : D \to \mathbb{R}$ and $v : D \to \mathbb{R}$ with $D \subseteq \mathbb{R}$. Then $\mathcal{W} = \mathcal{U} \circ \mathcal{V}$ defined by rules (8), (9), and (10) of the slope arithmetic is a slope triple of the function $w = u \circ f$, where $\circ \in \{+, -, \cdot, /\}$ and $0 \notin V_x$ if $\circ = /$. Also, $\mathcal{W} = \varphi(\mathcal{V})$ defined by rule (11) of the slope arithmetic is a slope triple of the function $w = \varphi(u)$, where φ is an elementary function.*

Proof: Let $x \in X$ and $c \in X$, then for all $\circ \in \{+, -, \cdot, /\}$,

$$\begin{aligned} w(x) &= u(x) \circ v(x) \in U_x \circ V_x = W_x, \\ w(c) &= u(c) \circ v(c) \in U_c \circ V_c = W_c. \end{aligned}$$

For elementary functions φ,

$$\begin{aligned} w(x) &= \varphi(u(x)) \in \varphi(U_x) = W_x, \\ w(c) &= \varphi(u(c)) \in \varphi(U_c) = W_c. \end{aligned}$$

Moreover, for $w = u \pm v$,

$$\begin{aligned} w(x) - w(c) &= u(x) \pm v(x) - (u(c) \pm v(c)) \\ &= (u(x) - u(c)) \pm (v(x) - v(c)) \\ &= s_u(c, x) \cdot (x - c) \pm s_v(c, x) \cdot (x - c) \\ &= (s_u(c, x) \pm s_v(c, x)) \cdot (x - c) \\ &\in \underbrace{(U_s \pm V_s)}_{= W_s} \cdot (x - c). \end{aligned}$$

For $w = u \cdot v$,

$$\begin{aligned} w(x) - w(c) &= u(x) \cdot v(x) - u(c) \cdot v(c) \\ &= u(x) \cdot v(x) - u(x) \cdot v(c) + u(x) \cdot v(c) - u(c) \cdot v(c) \\ &= u(x) \cdot (v(x) - v(c)) + (u(x) - u(c)) \cdot v(c) \\ &= u(x) \cdot s_v(c,x) \cdot (x-c) + s_u(c,x) \cdot (x-c) \cdot v(c) \\ &= (u(x) \cdot s_v(c,x) + s_u(c,x) \cdot v(c)) \cdot (x-c) \\ &\in \underbrace{(U_x \cdot V_s + U_s \cdot V_c)}_{= W_s} \cdot (x-c). \end{aligned}$$

For $w = u/v$,

$$\begin{aligned} w(x) - w(c) &= u(x)/v(x) - u(c)/v(c) \\ &= (u(x) - u(c) + u(c) - \frac{u(c)}{v(c)} \cdot v(x))/v(x) \\ &= (u(x) - u(c) + u(c) - w(c) \cdot v(x))/v(x) \\ &= (u(x) - u(c) - w(c) \cdot (v(x) - v(c)))/v(x) \\ &= (s_u(c,x) \cdot (x-c) - w(c) \cdot s_v(c,x) \cdot (x-c))/v(x) \\ &= \frac{s_u(c,x) - w(c) \cdot s_v(c,x)}{v(x)} \cdot (x-c) \\ &\in \underbrace{\frac{U_s - W_c \cdot V_s}{V_x}}_{= W_s} \cdot (x-c). \end{aligned}$$

For $w = \varphi(u)$,

$$\begin{aligned} w(x) - w(c) &= \varphi(u(x)) - \varphi(u(c)) \\ &= s_\varphi(u(c), u(x)) \cdot (u(x) - u(c)) \\ &= s_\varphi(u(c), u(x)) \cdot s_u(c,x) \cdot (x-c) \\ &\in \underbrace{s_\varphi(U_c, U_x) \cdot U_s}_{= W_s} \cdot (x-c). \end{aligned}$$

□

In practice, $s_\varphi(U_c, U_x)$ is not computed exactly in most cases. Usually, an enclosure $S_\varphi \supset s_\varphi(U_c, U_x)$ of the range of the slope (i.e. an overestimation) is used instead, and usually $S_\varphi = \varphi'(U_x)$ is used for this purpose. But for some of the elementary functions the slope $s_\varphi(U_c, U_x)$ can be computed explicitely, which yields better (sharper) enclosures. Assuming W_x and W_c to be computed as given in rule (11), we use the following formulas to compute good slope enclosures:

$\mathcal{W} = \varphi(\mathcal{U}) = \mathcal{U}^2$:

$$S_\varphi = s_\varphi(U_c, U_x) = U_x + U_c \tag{12}$$

$\mathcal{W} = \varphi(\mathcal{U}) = \sqrt{\mathcal{U}}$:

$$S_\varphi = s_\varphi(U_c, U_x) = 1/(W_x + W_c) \tag{13}$$

$\mathcal{W} = \varphi(\mathcal{U}) = \exp(\mathcal{U})$:

$$S_\varphi = \begin{cases} \left[\dfrac{\underline{\underline{w_x}} - \underline{\underline{w_c}}}{\underline{u_x} - \underline{u_c}}, \dfrac{\overline{\overline{w_x}} - \overline{\overline{w_c}}}{\overline{u_x} - \overline{u_c}} \right] & \text{if } \underline{u_x} \neq \underline{u_c} \ \wedge \ \overline{u_x} \neq \overline{u_c} \\ W_x & \text{otherwise} \end{cases} \tag{14}$$

$\mathcal{W} = \varphi(\mathcal{U}) = \ln(\mathcal{U})$:

$$S_\varphi = \begin{cases} \left[\dfrac{\overline{\overline{w_x}} - \overline{\overline{w_c}}}{\overline{u_x} - \overline{u_c}}, \dfrac{\underline{\underline{w_x}} - \underline{\underline{w_c}}}{\underline{u_x} - \underline{u_c}} \right] & \text{if } \underline{u_x} \neq \underline{u_c} \ \wedge \ \overline{u_x} \neq \overline{u_c} \\ 1/U_x & \text{otherwise} \end{cases} \tag{15}$$

$\mathcal{W} = \varphi(\mathcal{U}) = \mathcal{U}^k$, $k > 2$:

$$S_\varphi = \begin{cases} \left[\dfrac{\underline{\underline{u_x}^k} - \underline{\underline{u_c}^k}}{\underline{u_x} - \underline{u_c}}, \dfrac{\overline{\overline{u_x}^k} - \overline{\overline{u_c}^k}}{\overline{u_x} - \overline{u_c}} \right] & \text{if } k \text{ even } \wedge \ \underline{u_x} \neq \underline{u_c} \ \wedge \ \overline{u_x} \neq \overline{u_c} \\ \left[\dfrac{\underline{\underline{w_x}} - \underline{\underline{w_c}}}{\underline{u_x} - \underline{u_c}}, \dfrac{\overline{\overline{w_x}} - \overline{\overline{w_c}}}{\overline{u_x} - \overline{u_c}} \right] & \text{if } k \text{ odd } \wedge \ \underline{u_x} \geq 0 \ \wedge \ \underline{u_x} \neq \underline{u_c} \ \wedge \ \overline{u_x} \neq \overline{u_c} \\ \left[\dfrac{\overline{\overline{w_x}} - \overline{\overline{w_c}}}{\overline{u_x} - \overline{u_c}}, \dfrac{\underline{\underline{w_x}} - \underline{\underline{w_c}}}{\underline{u_x} - \underline{u_c}} \right] & \text{if } k \text{ odd } \wedge \ \overline{u_x} \leq 0 \ \wedge \ \underline{u_x} \neq \underline{u_c} \ \wedge \ \overline{u_x} \neq \overline{u_c} \\ k(U_x)^{k-1} & \text{otherwise} \end{cases} \tag{16}$$

To prove that Formulas (12) to (16) give enclosures for the true interval slopes, we need the following

Lemma 3.4 *Let $\varphi : D \to I\!R$ continously differentiable on $D \supset X \in I\!I\!R$, $C \overset{\circ}{\subset} X$. If φ is convex on X, then*

$$s_\varphi(\underline{c}, \underline{x}) \leq s_\varphi(c, x) \leq s_\varphi(\overline{c}, \overline{x}) \qquad \forall c \in C \ \ \forall x \in X, \ \ x \neq c.$$

If φ is concave on X, then

$$s_\varphi(\underline{c}, \underline{x}) \geq s_\varphi(c, x) \geq s_\varphi(\overline{c}, \overline{x}) \qquad \forall c \in C \ \ \forall x \in X, \ \ x \neq c.$$

Proof: If φ is convex on X, then for all $a, b \in X$ we have

$$\varphi(a) \geq \varphi(b) + (a - b)\varphi'(b).$$

for all $a, b \in X$. Therefore, we have

$$\begin{aligned} \frac{\partial}{\partial x}(s_\varphi(c, x)) &= \frac{\partial}{\partial x}\left(\frac{\varphi(x) - \varphi(c)}{x - c}\right) \\ &= \frac{\varphi'(x)(x - c) - \varphi(x) + \varphi(c)}{(x - c)^2} \\ &\geq 0 \end{aligned}$$

and

$$\begin{aligned}\frac{\partial}{\partial c}(s_\varphi(c,x)) &= \frac{\partial}{\partial c}\left(\frac{\varphi(x)-\varphi(c)}{x-c}\right)\\ &= \frac{-\varphi'(c)(x-c)+\varphi(x)-\varphi(c)}{(x-c)^2}\\ &\geq 0.\end{aligned}$$

Thus, s_φ is monotonously increasing in x and in c. The proof of the concave case is analogous. □

Theorem 3.5 *Formulas (12) to (16) satisfy*

$$s_\varphi(u_c,u_x)\in S_\varphi \qquad \forall u_c\in U_c \;\; \forall u_x\in U_x, \quad u_x\neq u_c. \tag{17}$$

Proof: For $\mathcal{W}=\varphi(\mathcal{U})=\mathcal{U}^2$,

$$s_\varphi(u_c,u_x)=\frac{u_x^2-u_c^2}{u_x-u_c}=u_x+u_c\in U_x+U_c=S_\varphi,$$

which proves (17) for formula (12).
For $\mathcal{W}=\varphi(\mathcal{U})=\sqrt{\mathcal{U}}$,

$$s_\varphi(u_c,u_x)=\frac{\sqrt{u_x}-\sqrt{u_c}}{u_x-u_c}=\frac{1}{\sqrt{u_x}+\sqrt{u_c}}\in\frac{1}{W_x+W_c}=S_\varphi,$$

which proves (17) for formula (13).
For $\mathcal{W}=\varphi(\mathcal{U})=\exp(\mathcal{U})$, we know from Lemma 3.4 that

$$s_\varphi(u_c,u_x)\in[s_\varphi(\underline{u_c},\underline{u_x}),s_\varphi(\overline{u_c},\overline{u_x})]$$

if $\underline{u_x}\neq\underline{u_c}\;\wedge\;\overline{u_x}\neq\overline{u_c}$, and from (4) we know that $s_\varphi(u_c,u_x)\in\varphi'(U_x)$ in the remaining case, which proves (17) for formula (14).
For $\mathcal{W}=\varphi(\mathcal{U})=\ln(\mathcal{U})$, we know from Lemma 3.4 that

$$s_\varphi(u_c,u_x)\in[s_\varphi(\overline{u_c},\overline{u_x}),s_\varphi(\underline{u_c},\underline{u_x})]$$

if $\underline{u_x}\neq\underline{u_c}\;\wedge\;\overline{u_x}\neq\overline{u_c}$, and from (4) we know that $s_\varphi(u_c,u_x)\in\varphi'(U_x)$ in the remaining case, which proves (17) for formula (15).
For $\mathcal{W}=\varphi(\mathcal{U})=\mathcal{U}^k$, $k>2$, the proofs of the first two cases of (16) correspond to the proof of (14) since φ is convex in U_x, and the proof of the third case corresponds to those of (15), since φ is concave in U_x. The remaining case again follows from (4). □

Remarks: (i) Formulas (14) and (15) also apply to other convex and concave functions, respectively.
(ii) For functions which are only locally convex the techniques can also be used by case distinctions.

(iii) The inverse hyperbolic functions can also be expressed via the functions ln and $\sqrt{}$ and the arithmetic operations. For example we could use $\operatorname{arsinh}(x) = \ln(x + \sqrt{x^2+1})$.

We are now able to evaluate a function $f : D \to I\!R$ over an interval $A \in I\!I\!R$ with fixed $c \in A$ in our interval slope arithmetic delivering

$$f(\mathcal{X}) = f((A, c, 1)) = (Y_x, Y_c, Y_s)$$

and we have

$$f_{\mathrm{rg}}(A) \subseteq Y_x, \qquad f(c) \in Y_c, \qquad \text{and} \qquad f(x) - f(c) \in Y_s \cdot (x - c) \quad \forall x \in A.$$

Example 3.6 Let $f(x) = x^2 - 4x + 2$. Using the slope arithmetic, we compute the enclosure Y_s of $s_f(c, A)$ for $A = [1, 7]$ and $c = 4$ by

$$\begin{aligned} f(\mathcal{X}) &= f((A, c, 1)) \\ &= (A, c, 1)^2 - 4 \cdot (A, c, 1) + 2 \\ &= ([1, 7], 4, 1)^2 - (4, 4, 0) \cdot ([1, 7], 4, 1) + (2, 2, 0) \\ &= ([1, 49], 16, [5, 11]) - ([4, 28], 16, 4) + (2, 2, 0) \\ &= ([-25, 47], 2, [1, 7]) \\ &= (Y_x, Y_c, Y_s), \end{aligned}$$

and we have $Y_s = s_f(c, A) = [1, 7]$. In contrast, if we compute the interval evaluation of $f'(x) = 2x - 4$ over $A = [1, 7]$ (which might also be done by automatic differentiation [4]), we get

$$F'(A) = 2 \cdot [1, 7] - 4 = [-2, 10].$$

Now, if we compare the naive interval evaluation of f over A with the derivative and the slope extension we have

$$\begin{aligned} F(A) = A^2 - 4A + 2 &= [-25, 47], \\ f(c) + F'(A) \cdot (A - c) &= [-28, 32], \\ f(c) + Y_s \cdot (A - c) &= [-19, 23], \end{aligned}$$

underlining that the slope extension gives the best result.

4 Implementation

Using formulas (14) and (15) in an implementation we must be careful with the necessary roundings, since the computed slope enclosures must be guaranteed enclosures of the true slope intervals. Thus all lower bounds must be computed with rounding downwards and all upper bounds with rounding upwards, respectively.

In the following, we present an implementation of our slope arithmetic in the scientific programming language PASCAL–XSC [7] using the environment described in [3]. The module *slopes* supplies type definition, operators and elementary functions for our interval slope arithmetic. The procedure *fsEval* simplifies the mechanism of function evaluating. For a function of type *SlopeType*, *fsEval* computes and delivers the enclosures Y_x, Y_c, and Y_s.

```
{------------------------------------------------------------------------------}
{ Purpose: Definition of an interval slope arithmetic which allows function  }
{    evaluation with automatic differentiation for interval slopes.          }
{ Method: Overloading of operators and elementary functions for operations   }
{    of data type 'SlopeType'.                                               }
{ Global types, operators, functions, and procedures:                        }
{    type       SlopeType         : data type for interval slope arithmetic  }
{    operators  +, -, *, /        : operators of interval slope arithmetic   }
{    functions  SlopeConst,                                                  }
{               SlopeVar          : to define slope constants/variables      }
{    functions  fxValue,                                                     }
{               fcValue,                                                     }
{               fsValue           : to get function and slope values         }
{    functions  sqr, sqrt, power,                                            }
{               exp, sin, cos,... : elementary functions of slope arithmetic }
{    procedure  fsEval(...)       : to compute function and slope values     }
{------------------------------------------------------------------------------}
module slopes;

use
  i_ari, i_util;

{------------------------------------------------------------------------------}
{ Global type definition                                                       }
{------------------------------------------------------------------------------}
global type
  SlopeType = record fx, fc, fs : interval; end;

{------------------------------------------------------------------------------}
{ Transfer functions for constants and variables                               }
{------------------------------------------------------------------------------}
global function SlopeConst (c: real) : SlopeType;        { Generate constant }
begin                                                    {-------------------}
  SlopeConst.fx := c;
  SlopeConst.fc := c;
  SlopeConst.fs := 0;
end;

global function SlopeConst (c: interval) : SlopeType;    { Generate constant }
begin                                                    {-------------------}
  SlopeConst.fx := c;
  SlopeConst.fc := c;
  SlopeConst.fs := 0;
end;

global function SlopeVar (v: real) : SlopeType;          { Generate variable }
begin                                                    {-------------------}
  SlopeVar.fx := v;
  SlopeVar.fc := v;
  SlopeVar.fs := 1;
end;

global function SlopeVar (v: interval; c: real) : SlopeType; { Gen. variable }
begin                                                        {---------------}
  SlopeVar.fx := v;
  SlopeVar.fc := c;
  SlopeVar.fs := 1;
end;

{------------------------------------------------------------------------------}
{ Access functions for function and slope values                               }
{------------------------------------------------------------------------------}
global function fxValue (u: SlopeType) : interval;      { Get function value }
```

```
begin                                                    {--------------------}
  fxValue:= u.fx;
end;

global function fcValue (u: SlopeType) : interval;{ Get function value for c }
begin                                              {--------------------------}
  fcValue:= u.fc;
end;

global function fsValue (u: SlopeType) : interval;         { Get slope value }
begin                                                      {-----------------}
  fsValue:= u.fs;
end;

{-------------------------------------------------------------------------------}
{ Monadic operators + and - for SlopeType operands                              }
{-------------------------------------------------------------------------------}
global operator + (u: SlopeType) res: SlopeType;
begin
  res:= u;
end;

global operator - (u: SlopeType) res: SlopeType;
begin
  res.fx := -u.fx;
  res.fc := -u.fc;
  res.fs := -u.fs;
end;

{-------------------------------------------------------------------------------}
{ Operators +, -, *, and / for two SlopeType operands                           }
{-------------------------------------------------------------------------------}
global operator + (u,v: SlopeType) res: SlopeType;
begin
  res.fx := u.fx + v.fx;
  res.fc := u.fc + v.fc;
  res.fs := u.fs + v.fs;
end;

global operator - (u,v: SlopeType) res: SlopeType;
begin
  res.fx := u.fx   - v.fx;
  res.fc := u.fc   - v.fc;
  res.fs := u.fs - v.fs;
end;

global operator * (u,v: SlopeType) res: SlopeType;
begin
  res.fx := u.fx*v.fx;
  res.fc := u.fc*v.fc;
  res.fs := u.fs*v.fc + u.fx*v.fs;
end;

global operator / (u,v: SlopeType) res: SlopeType;
var hx: interval;
begin
  res.fx := u.fx/v.fx;
  hx := u.fc/v.fc;
  res.fc := hx;
  res.fs := (u.fs - hx*v.fs)/v.fx;
end;

{-------------------------------------------------------------------------------}
```

```
{ Operators +, -, *, and / for one interval and one SlopeType operand        }
{-------------------------------------------------------------------------}
global operator + (u: interval; v: SlopeType) res: SlopeType;
begin
  res.fx := u + v.fx;
  res.fc := u + v.fc;
  res.fs := v.fs;
end;

global operator - (u: interval; v: SlopeType) res: SlopeType;
begin
  res.fx := u - v.fx;
  res.fc := u - v.fc;
  res.fs := - v.fs;
end;

global operator * (u: interval; v: SlopeType) res: SlopeType;
begin
  res.fx := u*v.fx;
  res.fc := u*v.fc;
  res.fs := u*v.fs;
end;

global operator / (u: interval; v: SlopeType) res: SlopeType;
var hx: interval;
begin
  res.fx := u/v.fx;
  hx := u/v.fc;
  res.fc := hx;
  res.fs := -hx*v.fs/v.fx;
end;

global operator + (u: SlopeType; v: interval) res: SlopeType;
begin
  res.fx := u.fx + v;
  res.fc := u.fc + v;
  res.fs := u.fs;
end;

global operator - (u: SlopeType; v: interval) res: SlopeType;
begin
  res.fx := u.fx - v;
  res.fc := u.fc - v;
  res.fs := u.fs;
end;

global operator * (u: SlopeType; v: interval) res: SlopeType;
begin
  res.fx := u.fx * v;
  res.fc := u.fc * v;
  res.fs := u.fs * v;
end;

global operator / (u: SlopeType; v: interval) res: SlopeType;
begin
  res.fx := u.fx / v;
  res.fc := u.fc / v;
  res.fs := u.fs / v;
end;

{-------------------------------------------------------------------------}
{ Operators +, -, *, and / for one real and one SlopeType operand          }
{-------------------------------------------------------------------------}
```

```
global operator + (u: real; v: SlopeType) res: SlopeType;
begin
  res := intval(u) + v;
end;

global operator - (u: real; v: SlopeType) res: SlopeType;
begin
  res := intval(u) - v;
end;

global operator * (u: real; v: SlopeType) res: SlopeType;
begin
  res := intval(u) * v;
end;

global operator / (u: real; v: SlopeType) res: SlopeType;
begin
  res := intval(u) / v;
end;

global operator + (u: SlopeType; v: real) res: SlopeType;
begin
  res := u + intval(v);
end;

global operator - (u: SlopeType; v: real) res: SlopeType;
begin
  res := u - intval(v);
end;

global operator * (u: SlopeType; v: real) res: SlopeType;
begin
  res := u * intval(v);
end;

global operator / (u: SlopeType; v: real) res: SlopeType;
begin
  res := u / intval(v);
end;

{------------------------------------------------------------------------------}
{ Elementary functions for SlopeType arguments                                 }
{------------------------------------------------------------------------------}
global function sqr (u: SlopeType) : SlopeType;
begin
  sqr.fx  := sqr(u.fx);
  sqr.fc  := sqr(u.fc);
  sqr.fs := (u.fx + u.fc) * u.fs;
end;

global function power (u: SlopeType; k: integer) : SlopeType;
var
  hx, hxi, hxs, hc, h1  : interval;
  i, s                  : real;
begin
  if (k = 0) then
    power:= SlopeConst(1)
  else if (k = 1) then
    power:= u
  else if (k = 2) then
    power:= sqr(u)
  else
    begin
```

```
      hxi := power(u.fx.inf,k); hxs := power(u.fx.sup,k);
      hx  := hxi +* hxs;
      if (not odd(k)) and (0 in u.fx) then
        hx.inf := 0;

      hc  := power(u.fc,k);
      i:= u.fx.inf - u.fc.inf;
      s:= u.fx.sup - u.fc.sup;
      if (i = 0) or (s = 0) or (odd(k) and (0 in u.fx)) then
        h1 := k * power(u.fx, k-1)
      else
        begin
          if not odd(k) then
            begin
              h1.inf := (hxi.sup -> hc.inf) /< pred(i);
              h1.sup := (hxs.sup -> hc.inf) /> pred(s);
            end
          else if u.fx.inf >= 0 then
            begin
              h1.inf := (hxi.sup -> hc.inf) /< pred(i);
              h1.sup := (hxs.sup -> hc.inf) /> pred(s);
            end
          else
            begin
              h1.inf := (hxs.inf -< hc.sup) /< succ(s);
              h1.sup := (hxi.inf -< hc.sup) /> succ(i);
            end
        end;
      power.fx := hx;
      power.fc := hc;
      power.fs := h1*u.fs;
    end;
end;

global function sqrt (u: SlopeType) : SlopeType;
var hx, hc: interval;
begin
  hx := sqrt(u.fx);
  hc := sqrt(u.fc);
  sqrt.fx := hx;
  sqrt.fc := hc;
  sqrt.fs := u.fs / (hx + hc);
end;

global function exp (u: SlopeType) : SlopeType;
var
  hxi, hxs, hci, hcs, i, s : real;
  h1                       : interval;
begin
  hxi := exp(u.fx.inf);  hxs := exp(u.fx.sup);
  hci := exp(u.fc.inf);  hcs := exp(u.fc.sup);
  exp.fx := intval(pred(hxi),succ(hxs));
  exp.fc := intval(pred(hci),succ(hcs));
  i:= u.fx.inf -< u.fc.inf;
  s:= u.fx.sup -< u.fc.sup;
  if (i = 0) or (s = 0) then
    h1 := intval(pred(hxi),succ(hxs))
  else
    begin
      h1.inf := (succ(hxi) -> pred(hci)) /< i;
      h1.sup := (succ(hxs) -> pred(hcs)) /> s;
    end;
  exp.fs := h1*u.fs;
```

```
end;

global function ln (u: SlopeType) : SlopeType;
var
  hxi, hxs, hci, hcs, i, s : real;
  h1                       : interval;
begin
  hxi := ln(u.fx.inf);  hxs := ln(u.fx.sup);
  hci := ln(u.fc.inf);  hcs := ln(u.fc.sup);
  ln.fx := intval(pred(hxi),succ(hxs));
  ln.fc := intval(pred(hci),succ(hcs));
  i:= u.fx.inf -> u.fc.inf;
  s:= u.fx.sup -> u.fc.sup;
  if (i = 0) or (s = 0) then
    h1 := 1/u.fx
  else
    begin
      h1.inf := (pred(hxs) -< succ(hcs)) /< s;
      h1.sup := (pred(hxi) -< succ(hci)) /> i;
    end;
  ln.fs := h1*u.fs;
end;

{-------------------------------------------------------------------------------}
{ Further elementary functions follow:                                          }
{     sin,  cos,  tan,  cot,  arcsin, arccos, arctan, arccot                    }
{     sinh, cosh, tanh, coth, arsinh, arcosh, artanh, arcoth                    }
{-------------------------------------------------------------------------------}

...


{-------------------------------------------------------------------------------}
{ Purpose: Evaluation of function 'f' for argument 'x' in interval slope        }
{    arithmetic computing enclosures of the function value, the midpoint        }
{    value, and the value of the slope.                                         }
{ Parameters:                                                                   }
{    In     : 'f'       : function of 'SlopeType'.                             }
{             'x', 'c'   : arguments for evaluation of 'f',                     }
{    Out    : 'fx'      : returns the function value 'f(x)'.                   }
{             : 'fc'     : returns the function value 'f(c)'.                   }
{             'fsx'      : returns the slope value 's(x;c)'.                    }
{-------------------------------------------------------------------------------}
global procedure fsEval (function f(x:SlopeType) : SlopeType;
                                                   x : interval;
                                                   c : real;
                                 var    fx, fc, fsx : interval);
var
  fxD : SlopeType;
begin
  fxD:= f(SlopeVar(x,c));
  fx:= fxD.fx;
  fc:= fxD.fc;
  fsx:= fxD.fs;
end;

{-------------------------------------------------------------------------------}
{ Module initialization part                                                    }
{-------------------------------------------------------------------------------}
begin
end.
```

5 Some Examples

In this section we give some examples comparing the slope enclosures and the corresponding extensions of our new optimized slope arithmetic with a usual slope arithmetic and a derivative arithmetic. We use the following functions:

1. $f(x) = (x + \sin x) \cdot \mathrm{e}^{-x^2}$ (taken from [13])
2. $f(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$
3. $f(x) = (\ln(x + 1.25) - 0.84x)^2$
4. $f(x) = \dfrac{2}{100}x^2 - \dfrac{3}{100}\mathrm{e}^{-(20(x-0.875))^2}$
5. $f(x) = \mathrm{e}^{x^2}$
6. $f(x) = x^4 - 12x^3 + 47x^2 - 60x - 20\mathrm{e}^{-x}$
7. $f(x) = x^6 - 15x^4 + 27x^2 + 250$ (taken from [5])

In Table 1 we list the enclosures for the slopes or derivatives (D, S_{old}, S_{new}) and in Table 1 we give the corresponding extension intervals (E_D, $E_{S_{\mathrm{old}}}$, $E_{S_{\mathrm{new}}}$) computed for the argument $X = [0.75, 1.75]$ and $c = m(X)$. We use the notations

$$\begin{array}{lll} D \supseteq F'(X) & \text{and} & E_D = f(c) + D(X - c), \\ S \supseteq s_f(c, X) & \text{and} & E_S = f(c) + S(X - c), \end{array}$$

and the additional subscripts *old* and *new*, indicating whether a usual slope arithmetic (using only formulas (12) and (13)) or our new one (using also formulas (14) to (16)) is applied. In the tables we use four significant digits and rounding outwards. The results underline the advantages of our new arithmetic.

Table 1: Computed derivatives and slopes for the sample functions

no.	D	S_{old}	S_{new}
1	[−5.446, 0.8863]	[−4.529, 0.2291]	[−2.800, 0.05215]
2	[−87.69, 77.07]	[−70.19, 59.57]	[−43.88, 38.26]
3	[−0.4749, 0.7873]	[−0.1697, 0.4614]	[−0.1592, 0.4329]
4	[−2.971, 21.08]	[0.03999, 15.07]	[0.03999, 0.3267]
5	[2.632, 74.84]	[3.5100, 64.15]	[6.031, 33.23]
6	[−94.59, 115.2]	[−71.09, 91.64]	[−39.00, 65.56]
7	[−279.7, 167.7]	[−266.2, 154.2]	[−146.9, 67.07]

Table 2: Computed extensions for the sample functions

no.	E_D	$E_{S_{\text{old}}}$	$E_{S_{\text{new}}}$
1	$[-2.262, 3.184]$	$[-1.804, 2.726]$	$[-0.9387, 1.861]$
2	$[-44.75, 42.95]$	$[-36.00, 34.20]$	$[-22.84, 21.04]$
3	$[-0.3758, 0.4115]$	$[-0.2128, 0.2486]$	$[-0.1986, 0.2343]$
4	$[-10.51, 10.57]$	$[-7.499,7.562]$	$[-0.1321, 0.1946]$
5	$[-32.65, 42.19]$	$[-27.31, 36.85]$	$[-11.84, 21.39]$
6	$[-85.86, 29.28]$	$[-74.11, 17.53]$	$[-61.07, 4.492]$
7	$[119.5, 399.3]$	$[126.3, 392.5]$	$[185.9, 332.9]$

6 An Application in Global Optimization

Interval branch-and-bound methods for global optimization address the problem of finding guaranteed and reliable solutions of global optimization problems

$$\min_{x \in X} f(x), \tag{18}$$

where $f : D \to \mathbb{R}$ is the objective function and $X \subseteq D$ is the search box representing bound constraints for x. These methods usually apply several interval techniques to reject regions which cannot contain the optimum. For this reason, the original box X gets subdivided, and subregions which cannot contain a global minimizer of f are discarded, while the other subregions get subdivided again until the desired accuracy (width) of the boxes is achieved.

Very often, if f is continuously differentiable, these interval methods incorporate the so called monotonicity test (see [2, 3, 5, 6, 9, 10], for example) to discard boxes. This test uses first-order information of the objective function by means of an interval evaluation of the derivative over the current box. Depending on this enclosure containing zero or not, the current box must be treated further or can be deleted, respectively.

As we have seen in the previous sections, interval slopes offer the possibility to achieve better enclosures for the function range. Thus, they might improve the performance of interval branch-and-bound methods. Although, since slopes cannot be used within the monotonicity test (see Section 6.1 for details), the need of a global optimization method with an alternative box-discarding technique arises. In this section, we describe the practical realization of such a method which incorporates a special pruning step generated by interval slopes. The theory of this pruning step is developed in [12]. It offers the possibility to cut away a large part of the current box, independently of the slope interval containing zero or not.

In the following, $X \subseteq D \subseteq \mathbb{R}$ and $f : D \to \mathbb{R}$. The global minimum value of f on X is denoted by f^*, and the set of global minimizer points of f on X by X^*. That is,

$$f^* = \min_{x \in X} f(x) \qquad \text{and} \qquad X^* = \{x^* \in X \mid f(x^*) = f^*\}.$$

6.1 A Pruning Technique Using Slopes

In first-order interval methods for global optimization, the monotonicity test determines whether the function f is *strictly monotone* within an entire subinterval $Y \subseteq X$. In this case Y cannot contain a global minimizer in its interior. Furthermore, a global minimizer can only lie on a boundary point of Y if this point is a boundary point of X as well. Therefore, if f satisfies

$$0 \notin F'(Y), \tag{19}$$

then the subinterval Y can be deleted (with the exception of boundary points of X).

If we want to apply slopes instead of derivatives, we cannot use this monotonicity test, since we have $s_f(c,X) \subseteq F'(X)$, but in general it is *not* true that $f'(x) \in s_f(c,X) \quad \forall c, x \in X$. Therefore, although $x^* \in Y$ is a local (or even global) minimizer with $f'(x^*) = 0$, it might happen that $0 \notin s_f(c,Y)$, and the latter cannot be used as a criterion to discard the box Y.

Example 6.1 We consider once more the function f from Example 3.6. Since $f(x) = x^2 - 4x + 2 = (x-2)^2 - 2$, we easily see that $x^* = 2$ is a local and global minimizer of f. With $Y = A = [1,7]$ we have $s_f(c,Y) = [1,7]$ showing that $0 \notin s_f(c,Y)$ cannot be used as a criterion to discard Y, since $x^* \in Y$.

On the other hand, it is underlined in [2], that the monotonicity test is an essential accelerating tool for an efficient interval global optimization method. Thus, the need of a corresponding tool in connection with slopes arises. It is called a *pruning step using slopes*, and its theory is developed in [12].

In this section we summarize the ideas and the theory presented in [12], and we give "implementation versions" of the theorems from [12], in which we take into account that, in general, $s_f(c,Y)$ and $f(c)$ cannot be computed exactly.

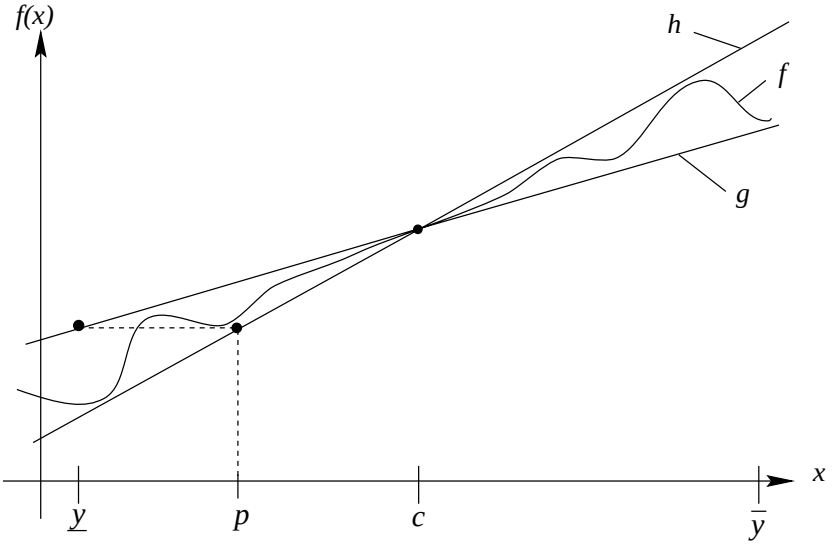


Figure 1: Generation of the pruning point p for positive interval slope

Figure 1 illustrates the idea for finding an upper bound p for all global minimizers within the interval Y if a slope interval $S = [\underline{s}, \overline{s}] = s_f(c, Y)$ with $\underline{s} > 0$ is known. First of all, we define the two lines

$$g : I\!R \to I\!R \qquad g(x) := f(c) + \underline{s} \cdot (x - c) \tag{20}$$

and

$$h : I\!R \to I\!R \qquad h(x) := f(c) + \overline{s} \cdot (x - c). \tag{21}$$

Then we know that $g(\underline{y})$ is an upper bound for $f(\underline{y})$ and thus for $\min_{x \in Y} f(x)$ in Y. Now we can locate p as the leftmost point in Y, for which f can not fall below $g(\underline{y})$. Since h is a lower bound for f in $[\underline{y}, c]$, we can do this very simply by computing the intersection point of h and the horizontal line r with $r(x) = g(\underline{y})$.

Usually, only enclosures for $s_f(c, Y)$ and $f(c)$ can be used in practical computations. Thus we state

Theorem 6.2 *Let $f : D \to I\!R$, $Y = [\underline{y}, \overline{y}] \in I\!I\!R$, $c \in Y \subseteq D \subseteq I\!R$, $f(c) \in Z = [\underline{z}, \overline{z}] \in I\!I\!R$. Moreover, let $S = [\underline{s}, \overline{s}] \supseteq s_f(c, Y)$ with $\underline{s} > 0$. Then*

$$p := c + ((\underline{y} - c) \cdot \underline{s} + d(Z))/\overline{s}$$

satisfies

$$\underline{y} \leq p \tag{22}$$

and

$$\min_{x \in Y} f(x) = \min_{x \in [\underline{y}, p]} f(x) < \min_{p < x \leq \overline{y}} f(x). \tag{23}$$

Proof: Since $\underline{y} \leq c$ and $0 < \underline{s}/\overline{s} \leq 1$, we have

$$\begin{aligned} \underline{y} \;=\; (1 - \underline{s}/\overline{s}) \cdot \underline{y} + \underline{s}/\overline{s} \cdot \underline{y} \;&\leq\; (1 - \underline{s}/\overline{s}) \cdot c + \underline{s}/\overline{s} \cdot \underline{y} \\ &=\; c + (\underline{y} - c) \cdot \underline{s}/\overline{s} \\ &\leq\; c + (\underline{y} - c) \cdot \underline{s}/\overline{s} + d(Z)/\overline{s} \;=\; p, \end{aligned}$$

which proves (22).

From (2) and (3) we know that for all $x \in (c, \overline{y}]$ there exists an $s_x > 0$ with $s_x \in S$ and

$$f(x) = f(c) + s_x \cdot (x - c).$$

Therefore, $f(x) > f(c) \quad \forall x \in (c, \overline{y}]$, and thus we know that

$$\min_{x \in Y} f(x) = \min_{x \in [\underline{y}, c]} f(x).$$

Now let $y^* \in Y$ with

$$f(y^*) = \min_{x \in Y} f(x). \tag{24}$$

If we assume that $p < y^* \le c$, then we know that there exist $s_l \in S$ and $s_* \in S$ satisfying $f(\underline{y}) = f(c) + s_l \cdot (\underline{y} - c)$ and $f(y^*) = f(c) + s_* \cdot (y^* - c)$. Thus we have

$$\begin{aligned} f(\underline{y}) \;=\; f(c) + s_l \cdot (\underline{y} - c) \;&\le\; \overline{z} + \underline{s} \cdot (\underline{y} - c) \\ &=\; \overline{z} + \underline{s} \cdot (((p - c) \cdot \overline{s} - d(Z))/\underline{s}) \\ &=\; \overline{z} + \overline{s} \cdot (p - c) - d(Z) \;=\; \underline{z} + \overline{s} \cdot (p - c) \\ &<\; \underline{z} + \overline{s} \cdot (y^* - c) \;\le\; \underline{z} + s_* \cdot (y^* - c) \\ &\le\; f(c) + s_* \cdot (y^* - c) \;=\; f(y^*), \end{aligned}$$

i.e. $f(\underline{y}) < f(y^*)$ which contradicts (24), and therefore $y^* \le p$ which proves (23). □

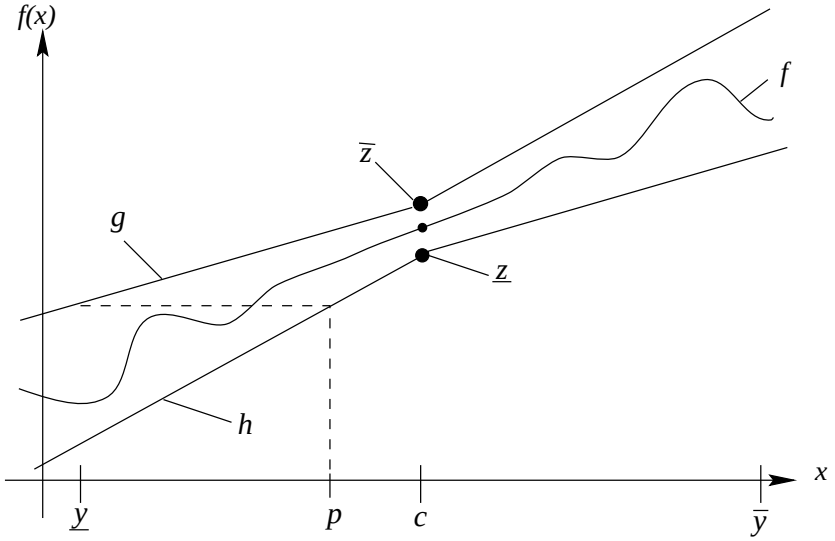


Figure 2: Generation of the pruning point p for $f(c) \in Z$

Figure 2 illustrates the situation treated in Theorem 6.2. Here, in contrast to Figure 1, the two lines

$$g : \mathbb{R} \to \mathbb{R} \qquad g(x) := \overline{z} + \underline{s} \cdot (x - c) \tag{25}$$

and

$$h : \mathbb{R} \to \mathbb{R} \qquad h(x) := \underline{z} + \overline{s} \cdot (x - c). \tag{26}$$

are used to generate the point p.

Using the value p of Theorem 6.2 within a global optimization method, we can prune a subinterval $Y \subseteq X$, if $0 < \underline{s} \le \overline{s}$ for $S \supseteq s_f(c, Y)$ to

$$Y_P := [\underline{y}, c + ((\underline{y} - c) \cdot \underline{s} + d(Z))/\overline{s}].$$

Example 6.3 We consider $f(x) = \frac{1}{2}x^2$, and we assume the current interval to be $Y = [-1, 4]$. First of all, we try to apply the monotonicity test. We evaluate the derivative $f'(x) = x$ over Y, and we get $F'(Y) = Y = [-1, 4]$. Since $0 \in F'(Y)$, we cannot discard Y from further consideration, and we must subdivide it and treat parts of Y in the same manner.

Now, we apply our new pruning step. We first evaluate the interval slope $S = s_f(c, Y) = \frac{1}{2}(c + Y)$, and with $c = 1.5$ we get $S = [0.25, 2.75]$. Since $0 \notin S$ we can prune Y to

$$Y_P = [\underline{y}, c + (\underline{y} - c) \cdot \underline{s}/\overline{s}] = [-1, 1.5 + (-1 - 1.5) \cdot 0.25/2.75] = [-1, 1.273]$$

using four significant digits and rounding outwards.

If we recall the situation in Figures 1 and 2, we see that we are able to improve the pruning of an interval Y. We can improve the point p (by moving it to the left), if we know a better (smaller) upper bound $\tilde{f}$ for $f(x)$ on Y than $g(\underline{y})$ was. Moreover, if $\tilde{f}$ is an upper bound for the global minimum value f^* on the whole search box X, then we can locate p as the leftmost point in Y, for which f can not fall below $\tilde{f}$. Since h is a lower bound for f near $\underline{y}$, we can do this by computing the intersection point of h and the horizontal line r with $r(x) = \tilde{f}$. In the context of a global optimization method using branch-and-bound techniques such as the cut-off test, an improved upper bound $\tilde{f}$ for the global minimum value f^* is usually known.

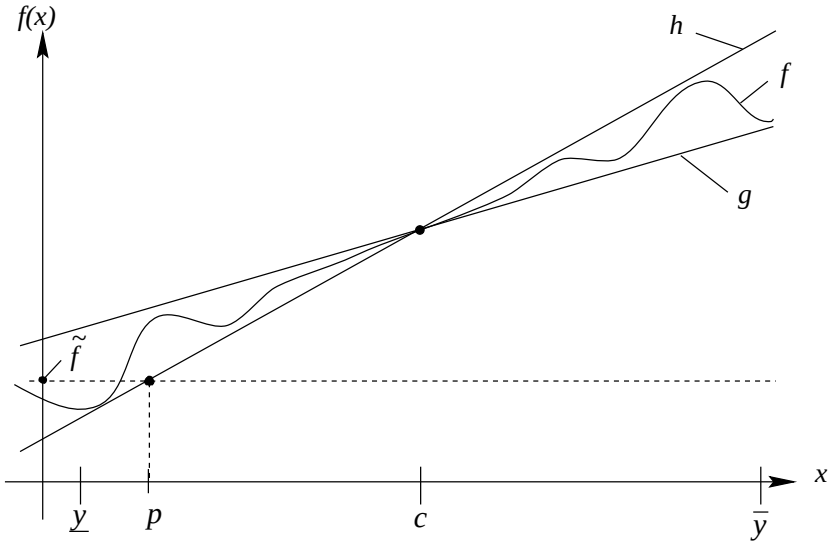


Figure 3: Generation of the pruning point p with known $\tilde{f}$

Figure 3 illustrates the idea for improving p when using the known upper bound $\tilde{f}$ for the global minimum value. Again we use the two lines

$$g : \mathbb{R} \to \mathbb{R} \qquad g(x) := f(c) + \underline{s} \cdot (x - c)$$

and

$$h : \mathbb{R} \to \mathbb{R} \qquad h(x) := f(c) + \overline{s} \cdot (x - c).$$

Then we know that $\tilde{f}$ is an upper bound for $\min_{x \in Y} f(x)$. Now we can locate p as the leftmost point in Y, for which f cannot fall below $\tilde{f}$. Since h is a lower bound for f in $[\underline{y}, c]$, we can do this very simply by computing the intersection point of h and the horizontal line r with $r(x) = \tilde{f}$.

Theorem 6.4 *Let $f : D \to I\!R$, $Y = [\underline{y}, \overline{y}] \in I\!I\!R$, $c \in Y \subseteq X \subseteq D \subseteq I\!R$, $f(c) \in Z = [\underline{z}, \overline{z}] \in I\!I\!R$. Moreover, let $S = [\underline{s}, \overline{s}] \supseteq s_f(c, Y)$ with $\underline{s} > 0$ and*

$$\widetilde{f} \geq f^* = \min_{x \in X} f(x). \tag{27}$$

Then $p := c + (m + d(Z))/\overline{s}$ with $m = \min\{\widetilde{f} - \overline{z},\ (\underline{y} - c) \cdot \underline{s}\}$ satisfies

$$\min_{p < x \leq \overline{y}} f(x) > f^* \quad \text{for } \underline{y} \leq p \tag{28}$$

or

$$\min_{x \in Y} f(x) > f^* \quad \text{for } p < \underline{y}, \tag{29}$$

respectively.

Proof: From (3) we know that for all $x \in (c, \overline{y}]$ there exists an $s_x > 0$ with $s_x \in S$ and

$$f(x) = f(c) + s_x \cdot (x - c).$$

Therefore, $f(x) > f(c) \quad \forall x \in (c, \overline{y}]$, which directly proves (28) and (29) for $p \geq c$.

Now, let $p < c$. For the case $m = (\underline{y} - c) \cdot \underline{s}$, Theorem 6.2 implies $\underline{y} \leq p$ and

$$\min_{p < x \leq \overline{y}} f(x) > \min_{x \in Y} f(x) \geq f^*.$$

For the case $m = \widetilde{f} - \overline{z}$, we assume that there exists an $x^* \in \mathcal{Z} := Y \cap (p, c]$ with $f(x^*) = f^*$. Then we know that there exists an $s_* \in S$ satisfying $f(x^*) = f(c) + s_* \cdot (x^* - c)$. Thus we have

$$\begin{aligned} f(x^*) \;=\; f(c) + s_* \cdot (x^* - c) \;&\geq\; \underline{z} + s_* \cdot (x^* - c) \;\geq\; \underline{z} + \overline{s} \cdot (x^* - c) \\ &> \underline{z} + \overline{s} \cdot (p - c) \\ &= \underline{z} + \overline{s} \cdot ((m + d(Z))/\overline{s}) \\ &= \underline{z} + m + \overline{z} - \underline{z} \;=\; \widetilde{f} - \overline{z} + \overline{z} \;=\; \widetilde{f}, \end{aligned}$$

which contradicts (27), and therefore $x^* \notin \mathcal{Z}$, which proves (28) and (29). □

So, we can use Theorem 6.4 within a global optimization method to prune or delete a subinterval $Y \subseteq X$, if $0 < \underline{s} \leq \overline{s}$ for $S = [\underline{s}, \overline{s}] \supseteq s_f(c, Y)$. That is, we first compute $Z = F(c)$ and

$$m = \min\{\widetilde{f} - \overline{z},\ (\underline{y} - c) \cdot \underline{s}\}$$

and then

$$p = c + (m + d(Z))/\overline{s}.$$

Then, if $p \geq \underline{y}$, we replace Y by

$$Y := [\underline{y}, p],$$

otherwise we delete the whole subbox Y.

It is easy to see, that we can apply a similar procedure for pruning in the case $\overline{s} < 0$ (cf. [12]). Moreover, the new pruning technique can succesfully be applied also in the

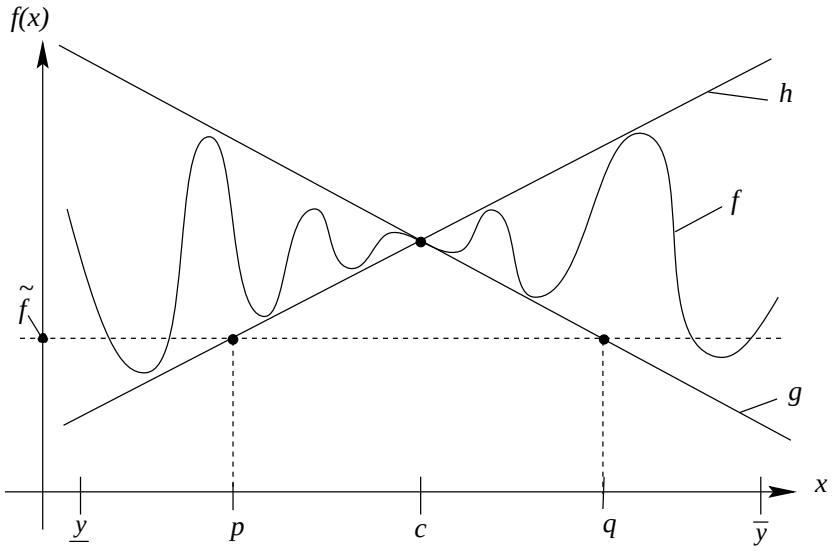


Figure 4: Generation of pruning points p and q with known $\tilde{f} < f(c)$

case $0 \in s_f(c, Y)$, which corresponds in a sense to the (unsuccessful) case $0 \in F'(Y)$ for the usual monotonicity test.

We illustrate this case in Figure 4. Again we use the two lines

$$g : I\!R \to I\!R \qquad g(x) := f(c) + \underline{s} \cdot (x - c)$$

and

$$h : I\!R \to I\!R \qquad h(x) := f(c) + \overline{s} \cdot (x - c),$$

assuming $\underline{s} < 0 < \overline{s}$. Now we can locate p as the leftmost point and q as the rightmost point in Y, for which f can not fall below $\tilde{f}$ according to the bounding by g and h. Since h is a lower bound for f in $[\underline{y}, c]$ and since g is a lower bound for f in $[c, \overline{y}]$, we can do this very simply by computing the intersection points of h and g with the horizontal line r with $r(x) = \tilde{f}$.

Theorem 6.5 *Let $f : D \to I\!R$, $Y = [\underline{y}, \overline{y}] \in I\!I\!R$, $c \in Y \subseteq X \subseteq D \subseteq I\!R$, $f(c) \in Z = [\underline{z}, \overline{z}] \in I\!I\!R$. Moreover, let $S = [\underline{s}, \overline{s}] \supseteq s_f(c, Y)$ with $0 \in S$ and*

$$\underline{z} > \tilde{f} \geq f^* = \min_{x \in X} f(x). \tag{30}$$

Then

$$p := \begin{cases} c + (\tilde{f} - \underline{z})/\overline{s} & \text{if } \overline{s} \neq 0, \\ -\infty & \text{otherwise,} \end{cases}$$

$$q := \begin{cases} c + (\tilde{f} - \underline{z})/\underline{s} & \text{if } \underline{s} \neq 0, \\ +\infty & \text{otherwise,} \end{cases}$$

and

$$\mathcal{Z} := \begin{cases} (p, q) \cap Y & \text{if } p < q, \\ \emptyset & \text{otherwise,} \end{cases}$$

satisfy

$$\min_{x \in \mathcal{Z}} f(x) > f^*. \tag{31}$$

Proof: We assume that there exists an $x^* \in \mathcal{Z} \neq \emptyset$ with $f(x^*) = f^*$. Then we know that there exists an $s_* \in S$ satisfying $f(x^*) = f(c) + s_* \cdot (x^* - c)$. In addition, we know that $x^* \neq c$, since equality would contradict (30).

If $x^* < c$, then

$$\begin{aligned} f(x^*) \;=\; f(c) + s_* \cdot (x^* - c) \;&\geq\; f(c) + \overline{s} \cdot (x^* - c) \\ &\geq\; \underline{z} + \overline{s} \cdot (x^* - c) \\ &>\; \underline{z} + \overline{s} \cdot (p - c) \\ &=\; \underline{z} + \overline{s} \cdot (\widetilde{f} - \underline{z})/\overline{s} \;=\; \widetilde{f}, \end{aligned}$$

which contradicts (30).

If $x^* > c$, then, in a similar way,

$$\begin{aligned} f(x^*) \;=\; f(c) + s_* \cdot (x^* - c) \;&\geq\; f(c) + \underline{s} \cdot (x^* - c) \\ &\geq\; \underline{z} + \underline{s} \cdot (x^* - c) \\ &>\; \underline{z} + \underline{s} \cdot (q - c) \\ &=\; \underline{z} + \underline{s} \cdot (\widetilde{f} - \underline{z})/\underline{s} \;=\; \widetilde{f}, \end{aligned}$$

which also contradicts (30).

Therefore $x^* \notin \mathcal{Z}$ and we proved (31). □

So, we can use Theorem 6.5 within a global optimization method to prune or delete a subinterval $Y \subseteq X$, if $\underline{s} \leq 0 \leq \overline{s}$ for $S \supseteq s_f(c, Y)$ and $f(c) \in Z$ and if $\widetilde{f} < \underline{z}$. That is, we first compute

$$p = c + (\widetilde{f} - \underline{z})/\overline{s} \quad \text{and} \quad q = c + (\widetilde{f} - \underline{z})/\underline{s}.$$

Then, we replace Y by

$$\begin{array}{ll} [\underline{y}, p] \cup [q, \overline{y}] & \text{if } \underline{y} \leq p \wedge q \leq \overline{y}, \\ [\underline{y}, p] & \text{if } \underline{y} \leq p \wedge q > \overline{y}, \\ [q, \overline{y}] & \text{if } \underline{y} > p \wedge q \leq \overline{y}, \end{array}$$

and otherwise we delete the whole subbox Y.

Remark: When implementing the pruning step on a machine, we must guarantee that all rounding errors are taken into account. Thus, in all three pruning cases the values p and q must be computed with upwardly-directed and downwardly-directed roundings, respectively.

6.2 Algorithmic Description of the Pruning Technique

We are now able to give an algorithmic formulation of a pruning step, which can be applied to a subinterval $Y \subseteq X$ when globally minimizing $f : D \to \mathbb{R}$ on $X \subseteq D$. The

algorithm uses

$$\begin{array}{rcl} Y & = & [\underline{y}, \overline{y}], \\ c & \in & Y, \\ Z & = & [\underline{z}, \overline{z}] \ \ni \ f(c), \\ S & = & [\underline{s}, \overline{s}] \ \supseteq \ s_f(c, Y), \quad \text{and} \\ \widetilde{f} & \geq & \min\limits_{x \in X} f(x) \end{array}$$

as input, and it delivers the pruned (and possibly empty) subset $U_1 \cup U_2$ of Y with $U_1, U_2 \in I\!I\!R \cup \{\emptyset\}$ and a possibly improved $\widetilde{f}$ as output. We use $\triangle(expr)$ and $\triangledown(expr)$ to indicate that an upper or lower bound for the expression $expr$ is computed.

Algorithm 6.1: SlopePruning $(Y, c, Z, S, \widetilde{f}, U_1, U_2)$

1. $U_1 := \emptyset; \quad U_2 := \emptyset;$
2. **if** $0 \in S$ **then** { pruning from the center }
3. **if** $\widetilde{f} < \underline{z}$ **then** { a pruning is possible }
4. **if** $\overline{s} > 0$ **then** { pruning from the center to the left }
5. $p := \triangle(c + (\widetilde{f} - \underline{z})/\overline{s});$
6. **if** $p \geq \underline{y}$ **then** $U_1 := [\underline{y}, p];$ { compute remaining left part }
7. **if** $\underline{s} < 0$ **then** { pruning from the center to the right }
8. $q := \triangledown(c + (\widetilde{f} - \underline{z})/\underline{s});$
9. **if** $q \leq \overline{y}$ **then** $U_2 := [q, \overline{y}];$ { compute remaining right part }
10. **else** { a pruning is *not* possible }
11. $U_1 := [\underline{y}, c]; \quad U_2 := [c, \overline{y}];$ { bisection }
12. **else if** $\underline{s} > 0$ **then** { pruning from right }
13. $\widetilde{f} := \min\{\widetilde{f}, \ \triangle((\underline{y} - c) \cdot \underline{s} + \overline{z})\};$ { update $\widetilde{f}$}
14. $p := \triangle(c + (\widetilde{f} - \underline{z})/\overline{s});$
15. **if** $p \geq \underline{y}$ **then** $U_1 := [\underline{y}, p];$ { compute remaining left part }
16. **else** { $\overline{s} < 0$ } { pruning from left }
17. $\widetilde{f} := \min\{\widetilde{f}, \ \triangle((\overline{y} - c) \cdot \overline{s} + \overline{z})\};$ { update $\widetilde{f}$}
18. $q := \triangledown(c + (\widetilde{f} - \underline{z})/\underline{s});$
19. **if** $q \leq \overline{y}$ **then** $U_2 := [q, \overline{y}];$ { compute remaining right part }
20. **return** $U_1, \ U_2, \ \widetilde{f};$

The following theorem summarizes the properties of this pruning step.

Theorem 6.6 *Let $f : D \to I\!R$, $Y \in I\!I\!R$, $c \in Y \subseteq X \subseteq D \subseteq I\!R$. Moreover, let $f(c) \in Z$, $s_f(c, Y) \subseteq S$, and $\widetilde{f} \geq \min_{x \in X} f(x)$, then Algorithm 6.1 applied as* SlopePruning $(Y, c, f_c, S, \widetilde{f}, U_1, U_2)$ *has the following properties:*

1. *$U_1 \cup U_2 \subseteq Y$.*
2. *Every global optimizer x^* of f in X with $x^* \in Y$ satisfies $x^* \in U_1 \cup U_2$.*
3. *If $U_1 \cup U_2 = \emptyset$, then there exists no global (w.r.t. X) optimizer of f in Y.*

Proof: Property 1 follows from the definition of U_1 and U_2. Theorems 6.4 and 6.5 directly imply Property 2. Property 3 is a consequence of Property 2. □

It is obvious that the success of Algorithm 6.1 in pruning Y depends on the quality of $\widetilde{f}$. Therefore, the pruning step within a global optimization method can very much benefit from a fast local search method delivering a good (small) value $\widetilde{f}$ on a very early stage of the method.

6.3 A Global Optimization Algorithm Using Pruning Steps

Subsequently, we give a simple first-order model algorithm where the pruning step is integrated. Our model algorithm uses the cut-off test, but it includes no local search procedure, no concavity test, and no Newton-like steps.

Algorithm 6.2: GlobalOptimize $(f, X, \varepsilon, F^*, L_{\rm res})$

```
1.  c := m(X);   f̃ := overline(F(c));                    { initialize upper bound }
2.  F_X := (F(c) + S_f(c,X)·(X − c)) ∩ F(X);              { centered form }
3.  L := {(X, underline(f_X))};   L_res := { };           { initialize working list and result list }
4.  while L ≠ { } do
5.      (Y, underline(f_Y)) := PopHead(L);   c := m(Y);   { get first element of working list }
6.      SlopePruning(Y, c, F(c), S_f(c,Y), f̃, U_1, U_2);
7.      for i := 1 to 2 do
8.          if U_i = ∅ then next_i;
9.          c := m(U_i);   if overline(F(c)) < f̃ then f̃ := overline(F(c));
10.         F_U := (F(c) + S_f(c,U_i)·(U_i − c)) ∩ F(U_i);   { centered form }
11.         if underline(f_U) ≤ f̃ then
12.             if d_rel(F_U) ≤ ε or d_rel(U_i) ≤ ε then
13.                 L_res := L_res ⊎ (U_i, underline(f_U))   { accept U_i for the result list }
14.             else
15.                 L := L ⊎ (U_i, underline(f_U));          { store U_i in the working list }
16.     endfor
17.     CutOffTest(L, f̃);
18. endwhile
19. (Y, underline(f_Y)) := Head(L_res);   F* := [underline(f_Y), f̃];   CutOffTest(L_res, f̃);
20. return F*,  L_res.
```

Algorithm 6.2 first computes an upper bound $\widetilde{f}$ for the global minimum value and initializes the working list L and the result list $L_{\rm res}$. The main iteration (from Step 4 to Step 18) starts with the pruning step applied to the leading interval of the working list. Then we apply a range check using a centered form to the resulting boxes U_1 and U_2 if they are non-empty. If the current box is still a candidate for containing a global minimizer, we store it in $L_{\rm res}$ (if it can be accepted with respect to the tolerance ε) or in L if it must be treated further.

Note that by the operation $\uplus$ the boxes are stored as pairs $(Y, \underline{f_Y})$ in list L sorted in *nondecreasing* order with respect to the lower bounds $\underline{f_Y} \le \underline{f_{\rm rg}(Y)}$ and in *decreasing*

order with respect to the ages of the boxes in L (cf. [11]). Thus, the leading box of L is the oldest element with the smallest $\underline{f_Y}$ value. When the iteration stops because the working list L is empty, we compute a final enclosure F^* for the global minimum value and return L_{res} and F^*.

The cut-off test is given by

Algorithm 6.3: CutOffTest $(L, \widetilde{f})$

1. **for all** $(Y, \underline{f_Y}) \in L$ **do**
2. **if** $\widetilde{f} < \underline{f_Y}$ **then** $L := L \uplus (Y, \underline{f_Y})$;
3. **endfor**
4. **return** L;

where $L \uplus (Y, \underline{f_Y})$ removes the element $(Y, \underline{f_Y})$ from L.

For our global optimization algorithm (Algorithm 6.2) we can state

Theorem 6.7 *Let $f : D \to \mathbb{R}$, $X \subseteq D \subseteq \mathbb{R}$, and $\varepsilon > 0$. Then Algorithm 6.2 has the following properties:*

1. $f^* \in F^*$.
2. $X^* \subseteq \bigcup_{(Y, \underline{f_Y}) \in L_{\mathrm{res}}} Y$.

Proof: Since the lists are sorted in non-decreasing order with respect to the $\underline{f_Y}$ values and since $\widetilde{f}$ is an upper bound of f^*, Assertion 1 is proved. Assertion 2 follows from the fact that neither the cut-off test nor the slope pruning step (due to Theorem 6.6) deletes boxes which contain a global minimizer of f. □

6.4 Some Examples and Tests

We implemented the global optimization algorithm (Algorithm 6.2) in PASCAL–XSC [7] Version 2.03. The test results presented in the following are generated on an HP 9000/730.

Example 6.8 To demonstrate the performance of our global optimization algorithm using pruning steps, we give an extract (about the first 9 steps) of the protocol of the pruning steps when applying Algorithm 6.2 on function

$$f(x) = \frac{(x-a)^2}{20} - \cos(x-a) + 2$$

with $a = 1.125$ and starting interval $X = [-5, 5]$.

For each current box Y in the while loop, we list its value, the value of the slope $S \supseteq s_f(c, Y)$, the chosen pruning step, and the resulting boxes U_1 and U_2. The empty set is represented by `[ / ]`.

```
    Y  = [ -5.000E+000,  5.000E+000 ]     S  = [ -1.363E+000,  1.138E+000 ]
==> bisection necessary
==> U1 = [ -5.000E+000,  0.000E+000 ]     U2 = [  0.000E+000,  5.000E+000 ]

    Y  = [  0.000E+000,  5.000E+000 ]     S  = [ -8.898E-001,  1.263E+000 ]
==> pruning by punching
==> U1 = [  0.000E+000,  2.288E+000 ]     U2 = [  2.801E+000,  5.000E+000 ]

    Y  = [  0.000E+000,  2.288E+000 ]     S  = [ -9.576E-001,  9.771E-001 ]
==> bisection necessary
==> U1 = [  0.000E+000,  1.144E+000 ]     U2 = [  1.143E+000,  2.288E+000 ]

    Y  = [  0.000E+000,  1.144E+000 ]     S  = [ -9.862E-001, -7.798E-003 ]
==> pruning from left
==> U1 = [           /              ]     U2 = [  7.384E-001,  1.144E+000 ]

    Y  = [  7.384E-001,  1.144E+000 ]     S  = [ -4.056E-001,  1.067E-002 ]
==> pruning by punching
==> U1 = [           /              ]     U2 = [  9.863E-001,  1.144E+000 ]

    Y  = [  9.863E-001,  1.144E+000 ]     S  = [ -1.481E-001,  1.687E-002 ]
==> pruning by punching
==> U1 = [           /              ]     U2 = [  1.077E+000,  1.144E+000 ]

    Y  = [  1.077E+000,  1.144E+000 ]     S  = [ -5.098E-002,  1.913E-002 ]
==> bisection necessary
==> U1 = [  1.077E+000,  1.111E+000 ]     U2 = [  1.110E+000,  1.144E+000 ]

    Y  = [  1.110E+000,  1.144E+000 ]     S  = [ -1.510E-002,  1.997E-002 ]
==> bisection necessary
==> U1 = [  1.110E+000,  1.128E+000 ]     U2 = [  1.127E+000,  1.144E+000 ]

    Y  = [  1.110E+000,  1.128E+000 ]     S  = [ -1.552E-002,  2.018E-003 ]
==> pruning by punching
==> U1 = [           /              ]     U2 = [  1.120E+000,  1.128E+000 ]
```

Extensive tests of our new slope pruning technique are documented in [12]. There the new method is compared with a corresponding method using the monotonicty test. According to these tests, our new method is always better than or at least as good as the traditional method with monotonicity test (with the exception of few cases where the list length was worse). On avarage, with the new pruning technique we have more than 30% improvement in the computation time and the number of function or derivative/slope evaluations. Moreover, there are many examples for which the required CPU time is reduced to around 1/3 of the time required by the variant with monotonicity test.

Now, we compare the two methods for two examples, for which we list the numerical results (the computed enclosures) and the evaluation efforts, the number of bisections, the necessary storage space, and the run-time.

Example 6.9 We minimize $f(x) = x^2/20 - \cos(x) + 2$.
Applying our model algorithm *using derivatives and monotonicity tests*, we get

```
Search interval         : [-20,20]
Tolerance (relative)    :     1E-8
No. of function calls   :      150
No. of derivative calls :       75
No. of bisections       :       37
Necessary list length   :        2
Run-time (in sec.)      :    0.700
Global minimizer in     : [-7.6293945312500E-005, 7.6293945312500E-005 ]
Global minimum value in : [ 1.0000000000000E+000, 1.0000000000001E+000 ]
```

Applying Algorithm 6.2 *using slopes and the new pruning steps*, we get

```
Search interval         : [-20,20]
Tolerance (relative)    :     1E-8
No. of function calls   :       58
No. of slope calls      :       29
No. of bisections       :        1
Necessary list length   :        2
Run-time (in sec.)      :    0.270
Global minimizer in     : [-7.4615903941273E-005, 7.4615903941273E-005 ]
Global minimum value in : [ 1.0000000000000E+000, 1.0000000000001E+000 ]
```

Example 6.10 We minimize Hansen's function $f(x) = 24x^4 - 142x^3 + 303x^2 - 276x + 93$ (cf. [5]).
Applying our model algorithm *using derivatives and monotonicity tests*, we get

```
Search interval         : [0,3]
Tolerance (relative)    :  1E-8
No. of function calls   :   956
No. of derivative calls :   478
No. of bisections       :   238
Necessary list length   :    25
Run-time (in sec.)      : 0.450
Global minimizer in     : [ 1.999990940E+000, 2.000009536E+000 ]
Global minimum value in : [ 9.999999982E-001, 1.000000001E+000 ]
```

Applying Algorithm 6.2 *using slopes and the new pruning steps*, we get

```
Search interval         : [0,3]
Tolerance (relative)    :  1E-8
No. of function calls   :   488
No. of slope calls      :   244
No. of bisections       :    12
Necessary list length   :    15
Run-time (in sec.)      : 0.180
Global minimizer in     : [ 1.999997019E+000, 2.000009781E+000 ]
Global minimum value in : [ 9.999999996E-001, 1.000000001E+000 ]
```

References

[1] Alefeld, G. and Herzberger, J. (1983): *Introduction to Interval Computations*, Academic Press, New York.

[2] Csendes, T. and Pintér, J. (1993): *The Impact of Accelerating Tools on the Interval Subdivision Algorithm for Global Optimization*, European J. of Operational Research, **65**, 314–320.

[3] Hammer, R., Hocks, M., Kulisch, U. and Ratz, D. (1993): *Numerical Toolbox for Verified Computing I*, Springer-Verlag, Berlin.

[4] Hammer, R., Hocks, M., Kulisch, U. and Ratz, D. (1995): *C++ Toolbox for Verified Computing I*, Springer-Verlag, Berlin.

[5] Hansen, E. (1992): *Global Optimization Using Interval Analysis*, Marcel Dekker, New York.

[6] Kearfott, R. B. (1996): *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers, Boston, 1996.

[7] Klatte, R., Kulisch, U., Neaga, M., Ullrich, Ch. and Ratz, D. (1992): *PASCAL–XSC – Language Description with Examples*, Springer-Verlag, Berlin.

[8] Neumaier, A. (1990): *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge.

[9] Ratschek, H. and Rokne, J. (1988): *New Computer Methods for Global Optimization*, Ellis Horwood, Chichester.

[10] Ratz, D. (1992): *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*, Dissertation, Universität Karlsruhe.

[11] Ratz, D. and Csendes, T. (1995): *On the Selection of Subdivision Directions in Interval Branch-and-Bound Methods for Global Optimization.* Journal of Global Optimization, **7**, 183–207.

[12] Ratz, D. (1997): *A New Global Optimization Technique Using Slopes – The One-Dimensional Case.* Submitted for publication in Journal of Global Optimization.

[13] Törn, A. and Žilinskas, A. (1989): *Global Optimization*, Lecture Notes in Computer Science, No. 350, Springer-Verlag, Berlin.

In dieser Reihe sind bisher die folgenden Arbeiten erschienen:

1/1996 Ulrich Kulisch: *Memorandum über Computer, Arithmetik und Numerik.*

2/1996 Andreas Wiethoff: *C–XSC — A C++ Class Library for Extended Scientific Computing.*

3/1996 Walter Krämer: *Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen.*

4/1996 Dietmar Ratz: *An Optimized Interval Slope Arithmetic and its Application.*

5/1996 Dietmar Ratz: *Inclusion Isotone Extended Interval Arithmetic.*

1/1997 Astrid Goos, Dietmar Ratz: *Praktische Realisierung und Test eines Verifikationsverfahrens zur Lösung globaler Optimierungsprobleme mit Ungleichungsnebenbedingungen.*

2/1997 Stefan Herbort, Dietmar Ratz: *Improving the Efficiency of a Nonlinear-System-Solver Using a Componentwise Newton Method.*

3/1997 Ulrich Kulisch: *Die fünfte Gleitkommaoperation für top-performance Computer — oder — Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik.*

4/1997 Ulrich Kulisch: *The Fifth Floating-Point Operation for Top-Performance Computers — or — Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic.*

5/1997 Walter Krämer: *Eine Fehlerfaktorarithmetik für zuverlässige a priori Fehlerabschätzungen.*